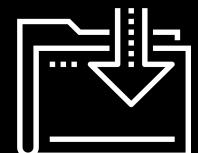




# GeoJSON & Leaflet Plugins

Data Boot Camp  
Lesson 17.2



# Class Objectives

---

By the end of today's class you will be able to:



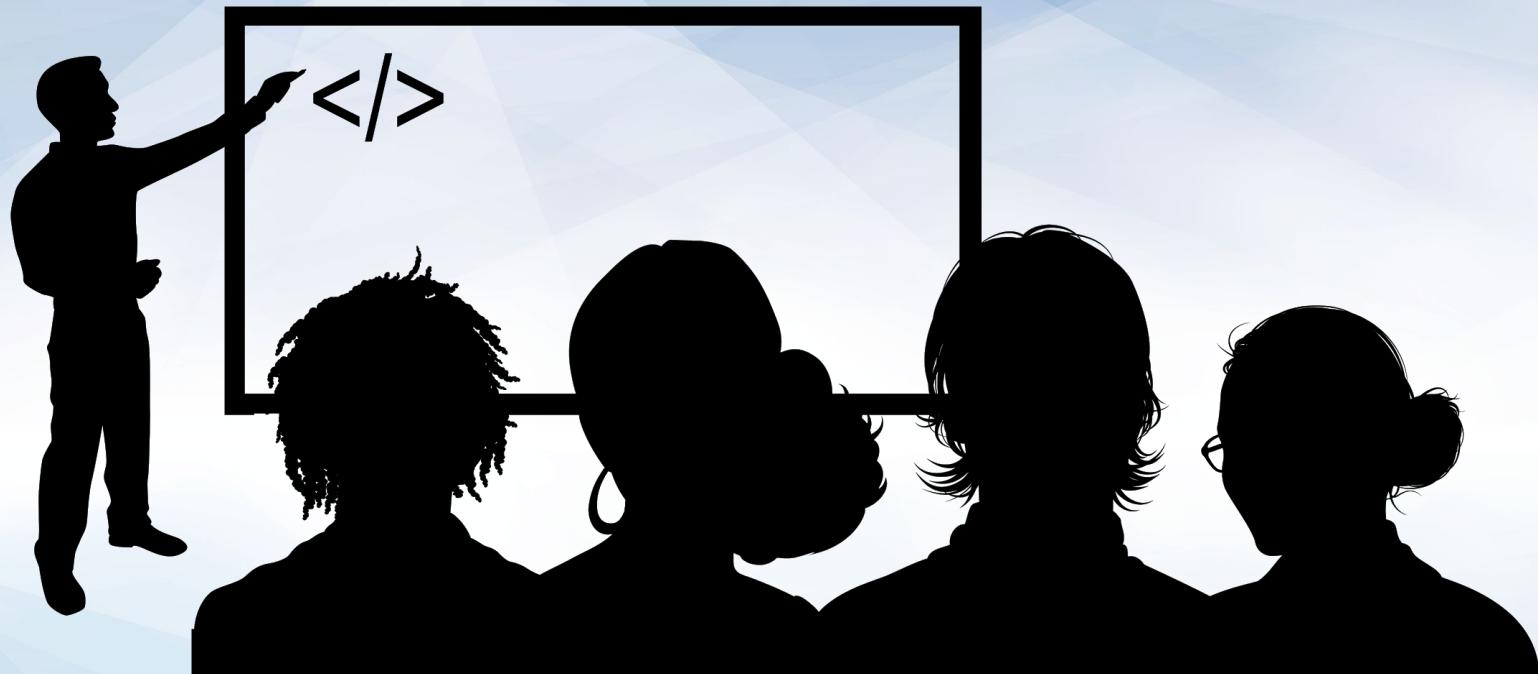
Gain a firm grasp of mapping with GeoJSON.



Learn about and practice using Leaflet plugins and third-party libraries.



Learn how different maps can effectively visualize different datasets.



## Instructor Demonstration GeoJSON Review



# What is GeoJSON?

→ Review  
**GeoJSON**

- **GeoJSON is a geospatial data interchange format based on JavaScript Object Notation (JSON). It defines several types of JSON objects and the manner in which they are combined to represent data about geographic features, their properties, and their spatial extents.**

# GeoJSON Review

```

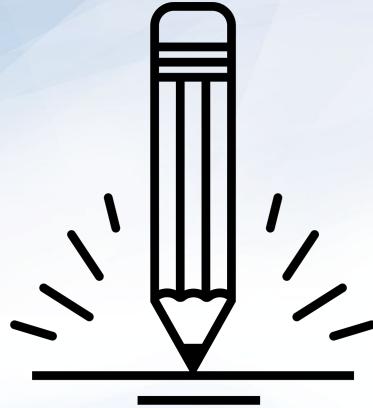
[{"type": "FeatureCollection", "metadata": {"id": "1403337170000", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/11_hour.geojson"}, "title": "USGS All Earthquakes, Past Hour", "status": "200", "api": "1.10.3", "version": "1.10.3", "features": [{"type": "Feature", "properties": {"mag": "1.29", "place": "13km SW of Seacres Valley, CA", "id": "16033344714783", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/c13940911", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a16033344714783", "status": "automatic", "tsunami": "no", "sig": 14, "net": "ci", "code": "60049911", "ids": "c13940911", "sources": "ci", "type": "nearby_cities,origin,phase,data,scisched"}, {"type": "Feature", "properties": {"mag": "1.5", "place": "50 km NW of Jiangyao, China", "id": "160333593083", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/us0000d41", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a160333593083", "status": "reviewed", "tsunami": "no", "sig": 400, "net": "us", "code": "60000041", "ids": "us0000d41", "sources": "us", "type": "origin,phase,data"}, {"type": "Feature", "properties": {"mag": "1.51", "place": "5.10 km NW of Jiangyao, China", "id": "16033344684040", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/us0000d41", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a16033344684040", "status": "reviewed", "tsunami": "no", "sig": 400, "net": "us", "code": "60000041", "ids": "us0000d41", "sources": "us", "type": "phase,data"}, {"type": "Feature", "properties": {"mag": "1.12", "place": "15km S of Trona, CA", "id": "160333449310", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/c139409085", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a160333449310", "status": "reviewed", "tsunami": "no", "sig": 19, "net": "ci", "code": "60000041", "ids": "c139409085", "sources": "ci", "type": "nearby_cities,origin,phase,data,scisched"}, {"type": "Feature", "properties": {"mag": "1.14", "place": "11-16 km E of Trona, CA", "id": "160333448995", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/c13940995", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a160333448995", "status": "reviewed", "tsunami": "no", "sig": 62, "net": "ci", "code": "60000041", "ids": "c13940995", "sources": "ci", "type": "phase,data"}, {"type": "Feature", "properties": {"mag": "1.15", "place": "15 km W of Ludlow, CA", "id": "1603334429420", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/c13940887", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a1603334429420", "status": "reviewed", "tsunami": "no", "sig": 62, "net": "ci", "code": "60000041", "ids": "c13940887", "sources": "ci", "type": "phase,data"}, {"type": "Feature", "properties": {"mag": "1.16", "place": "17-20 km SSE of Mina, Nevada", "id": "160333397220", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/m0779982", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a160333397220", "status": "reviewed", "tsunami": "no", "sig": 10, "net": "nn", "code": "60000041", "ids": "m0779982", "sources": "nn", "type": "phase,data"}, {"type": "Feature", "properties": {"mag": "1.47", "place": "Reykjanesskagi Ridge", "id": "160333400647", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/us0000cb7", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a160333400647", "status": "reviewed", "tsunami": "no", "sig": 340, "net": "us", "code": "60000047", "ids": "us0000cb7", "sources": "us", "type": "origin,phase,data"}, {"type": "Feature", "properties": {"mag": "2.0", "place": "5 km NW of Fritz Creek, Alaska", "id": "160333447282", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/us0000d47", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a160333447282", "status": "automatic", "tsunami": "no", "sig": 10, "net": "ak", "code": "6002016gk", "ids": "ak02016gk", "sources": "ak", "type": "origin,phase,data"}, {"type": "Feature", "properties": {"mag": "2.0", "place": "7 NW of Fritz Creek, Alaska", "id": "160333447284", "tsz": "null", "url": "https://earthquake.usgs.gov/eqevent/fed/v1/overall/us0000d47", "detail": "https://earthquake.usgs.gov/eqevent/fed/v1/detail/a160333447284", "status": "reviewed", "tsunami": "no", "sig": 84, "net": "gp", "code": "6002016gk", "ids": "gp02016gk", "sources": "gp", "type": "phase,data"}], "bbox": "170 885 890 899"}]

```

```
"type": "Feature",
  "properties": {
    "mag": 0.5,
    "place": "4km W of Cobb, California",
    "time": 1476329457770, ←
    "updated": "1476329552105",
    "tz": -420,
    "url": "http://earthquake.usgs.gov/earthquakes/eventpage/nc7271173",
    "detail": "http://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/1476329457770.geojson"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [
      -122.7771683, ←
      38.8195, ←
      0.35
    ]
  },
  "id": "nc72711736"
}, ←
```

When using GeoJSON with Leaflet, Leaflet expects each feature object to have a "geometry" property containing information about the type of marker that should be displayed and its coordinates.

GeoJSON may come with a "properties" object containing some metadata about the feature. In particular we are provided with some immediately useful information such as the place the earthquake occurred, the magnitude, and the time it was recorded.



## Everyone Do: NY Neighborhoods

In this activity, we will be diving into some advance Leaflet/GeoJSON functionality by building a map of New York City broken down by boroughs and neighborhoods.

**Suggested Time:**  
**20 Minutes**



# Everyone Do: NY Neighborhoods

## → File Structure

```
└ static
  └ CSS
    # style.css
  └ data
    nyc.geojson
  └ js
    JS config.js
    JS logic.js
    JS logic2.js
    JS logic3.js
    JS logic4.js
  <> index.html
```

The screenshot shows a comparison between a local file structure and a public dataset page on data.beta.nyc.

**File Structure:** On the left, a dark-themed file browser shows a directory structure. A red box highlights the file `nyc.geojson` under the `data` folder.

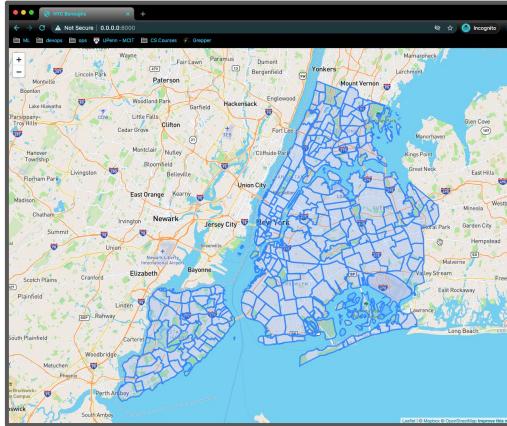
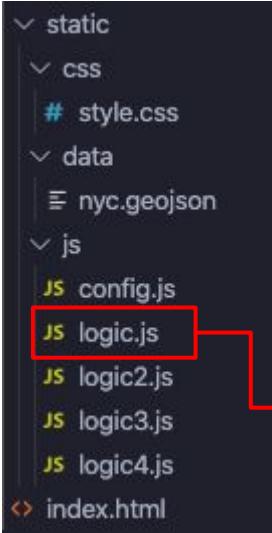
**Dataset Page:** On the right, a screenshot of the `data.beta.nyc` website displays the dataset `pediacities-nyc-neighborhoods`. The URL in the address bar is `data.beta.nyc/dataset/pediacities-nyc-neighborhoods`.

The page includes the following sections:

- Header:** `data.BetaNYC`, `DAT` logo, navigation links: DATASETS, SHOWCASES, ORGANIZATIONS, TOPICS, LOG IN, SIGN UP.
- Dataset Overview:** `NYC NEIGHBORHOODS`, `NYC Neighborhoods polygons and correlated data with their respective Postal Codes, Assembly Districts, Community Districts, Congressional Districts, Council Districts and State Senate Districts created by Ontodia.`
- Organization:** `datHere`, `DATHERE`, `Our Data Placed. Applied Urban Informatics using Open Infrastructure`, `read more`.
- Social:** `Twitter`, `Facebook`, `LinkedIn`.
- Data and Resources:** `pediacities-nyc-neighborhoods` (highlighted with a red box), `Polygon file of NYC Neighborhood boundaries maintained by Ontodia. Attribute table includes two...`, `Preview`, `Download`; `nyc_zip.borough_neighborhoods_pop.csv` (highlighted with a red box), `NYC zipcode with corresponding borough, neighborhood, population and density`, `Preview`, `Download`.

# Everyone Do: NY Neighborhoods

## → File Structure



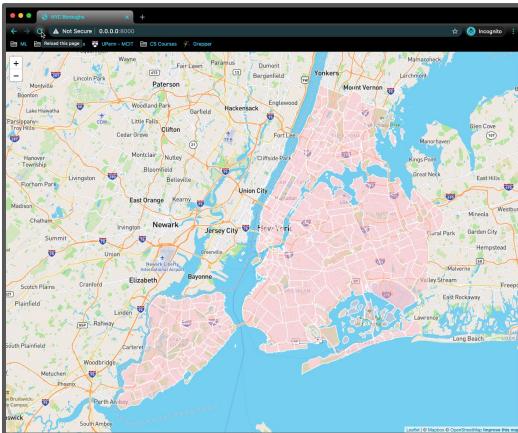
```
1 // Creating map object
2 var myMap = L.map("map", {
3   center: [40.7128, -74.0059],
4   zoom: 11
5 });
6
7 // Adding tile layer
8 L.tileLayer("https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}", {
9   attribution: "<a href='https://www.mapbox.com/about/maps/'>Mapbox</a> <a href='http://www.openstreetmap.org/copyright'>OpenStreetMap</a> <strong><a href='https://www.mapbox.com/map-feedback/'>Feedback</a></strong>",
10  tileSize: 512,
11  maxZoom: 18,
12  zoomOffset: -1,
13  id: "mapbox/streets-v11",
14  accessToken: API_KEY
15 }).addTo(myMap);
16
17 // Use this link to get the geojson data.
18 var link = "static/data/nyc.geojson";
19
20 // Grabbing our GeoJSON data..
21 d3.json(link, function(data) {
22   // Creating a GeoJSON layer with the retrieved data
23   L.geoJson(data).addTo(myMap);
24 })
```

- From the command line navigate to where `index.html` is and run:  
`python -m http.server`.
- Open <http://0.0.0.0:8000/> in a browser.

# Everyone Do: NY Neighborhoods

## → File Structure

```
└ static
  └ css
    # style.css
  └ data
    nyc.geojson
  └ js
    config.js
    logic.js
    logic2.js
    logic3.js
    logic4.js
  index.html
```



→ In the index.html file change to logic2.js

```
31 <!-- JS -->
32 <script type="text/javascript" src="static/js/logic2.js"></script>
33
```

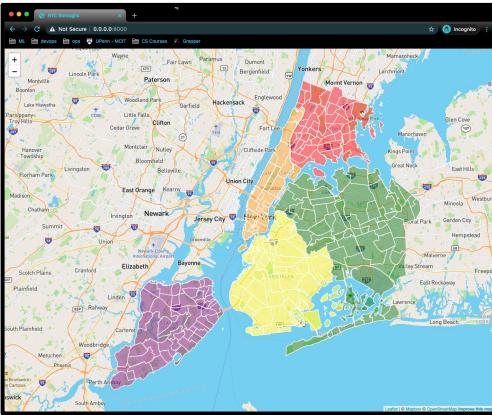
- From the command line navigate to where `index.html` is and run:  
`python -m http.server`.
- Open <http://0.0.0.0:8000/> in a browser.

```
1 // Creating map object
2 var myMap = L.map("map", {
3   center: [40.7128, -74.0059],
4   zoom: 11
5 });
6
7 // Adding tile layer
8 L.tileLayer("https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{y}?access_token={accessToken}", {
9   attribution: "© Mapbox</a> © OpenStreetMap</a> <strong><a href="https://www.mapbox.com/map-feedback/">Feedback</a></strong>",
10   tileSize: 512,
11   maxZoom: 18,
12   minZoom: 1,
13   id: "mapbox/streets-v11",
14   accessToken: API_KEY
15 }).addTo(myMap);
16
17 // Use this link to get the geojson data.
18 var link = "static/data/nyc.geojson";
19
20 // Our style object
21 var mapStyle = {
22   color: "white",
23   fillColor: "pink",
24   fillOpacity: 0.5,
25   weight: 15
26 };
27
28 // Grabbing our GeoJSON data.
29 d3.json(link, function(data) {
30   // Creating a geoJSON layer with the retrieved data
31   L.geoJson(data, {
32     style: mapStyle
33   }).addTo(myMap);
34 });
35
```

# Everyone Do: NY Neighborhoods

## → File Structure

```
└ static
  └ css
    # style.css
  └ data
    nyc.geojson
  └ js
    config.js
    logic.js
    logic2.js
    JS logic3.js
    JS logic4.js
  <> index.html
```



→ In the index.html file change to logic3.js

```
31     <!-- JS -->
32     <script type="text/javascript" src="static/js/logic3.js"></script>
33
```

- From the command line navigate to where `index.html` is and run:  
`python -m http.server`.
- Open <http://0.0.0.0:8000/> in a browser.

```
// Creating and object
var map = L.map("map").setView([40.7128, -74.0069], 11);
map.fitBounds(boroughs.getBounds());
map.on('click', function(e) {
  var borough = boroughs.getFeatureByContains(e.latlng);
  if (borough) {
    document.getElementById("info").innerHTML = "Borough: " + borough.properties.borough;
  }
});

// Adding tiles layer
L.tileLayer("https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}", {
  attribution: "Map data © 2016 Mapbox, © 2016 OpenStreetMap contributors. Imagery © 2016 Esri, DigitalGlobe, GeoEye, Intermap, Inpharos, Petrogeo, USDA FSA, USGS, Getmapping, Esri, DeLorme, HERE, and the GIS User Community",
  maxZoom: 18,
  zoomOffset: 1,
  id: "mapbox/streets-v11",
  accessToken: API_KEY
}).addTo(map);

// Using map to get the access data.
var boroughs = L.geoJson(data, {
  onEachFeature: onEachFeature
});

// Function that will determine the color of a neighborhood based on the borough it belongs to
function chooseColor(borough) {
  switch(borough) {
    case "Manhattan": return "#ffccbc";
    case "Brooklyn": return "#ff9800";
    case "Queens": return "#9467bd";
    case "Bronx": return "#ffbb78";
    case "Staten Island": return "#9e2e9c";
    default: return "#black";
  }
}

// Grabbing our geoJSON data...
$.getJSON(link, function(data) {
  // Grabbing our map layer and the data layer with the retrieved data
  L.geoJson(data, {
    style: chooseColor,
    onEachFeature: onEachFeature
  }).addTo(map);
  map.fitBounds(boroughs.getBounds());
  map.setZoom(10);
  map.setCenter([-74.0, 40.7]);
  map.setZoom(11);
  map.setZoom(11);
});
```

# Everyone Do: NY Neighborhoods

## → File Structure

```
└ static
  └ css
    # style.css
  └ data
    nyc.geojson
  └ js
    config.js
    logic.js
    logic2.js
    logic3.js
    logic4.js
```



↳ index.html

→ In the index.html file change to logic4.js

```
31   <!-- JS -->
32   <script type="text/javascript" src="static/js/logic4.js"></script>
```

- From the command line navigate to where index.html is and run:  
`python -m http.server`.
- Open <http://0.0.0.0:8000/> in a browser.

```
var myMap = L.map("map", {
  center: [40.7128, -74.006],
  zoom: 11
});

L.tileLayer("https://api.mapbox.com/v4/{id}/{z}/{x}/{y}.jpg?access_token={accessToken}", {
  id: "mapbox/streets-v11",
  tileSize: 512,
  maxZoom: 11,
  minZoom: 2,
  accessToken: "pk.eyJ1IjoiYXNkZWJhZG9tZWQyIiwiYSI6ImNqY2Fua2EwNzEwOTUifQ.1gDfPnVcOOGdWzvHrJLcA"
}).addTo(myMap);

// use this line to see the geojson data.
// var link = "static/nyc-geojson.json";
// Function that will determine the color of a neighborhood based on the borough it belongs to
function getBoroughColor(borough) {
  switch (borough) {
    case "Bronx": return "#58357E"; break;
    case "Brooklyn": return "#E65100"; break;
    case "Manhattan": return "#F08032"; break;
    case "Queens": return "#F0F5A7"; break;
    case "Staten Island": return "#8050A0"; break;
  }
  return "#A9A9A9";
}

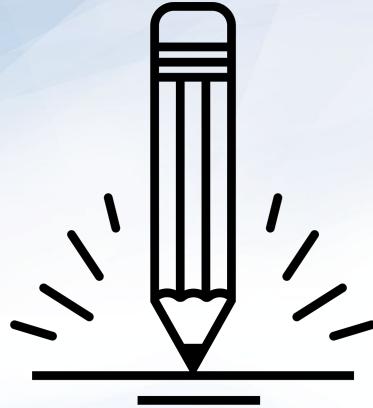
// Grabbing our geoJSON data...
$.getJSON("static/nyc-geojson.json", function(data) {
  // Add the geoJSON layer with the retrieved data
  L.geoJson(data, {
    // Set the color of each feature (this will color a neighborhood)
    style: function(feature) {
      return getBoroughColor(feature.properties.borough);
    },
    // Set the stroke width
    strokeWeight: 1.5
  }).addTo(myMap);
});

// Call on each feature
// When the user hovers over a map feature, layer {
//   // Get mouse events to change map styling
//   layer.on("mouseover", function() {
//     // when the user's mouse touches a map feature, the mouseover event calls this function, that Feature's opacity changes to 80% so that it stands out
//     feature.setStyle();
//   });
//   layer.on("mouseout", function() {
//     // when the user's mouse leaves a map feature, the mouseout event calls this function, that Feature's opacity reverts back to 50%
//     feature.setStyle();
//   });
// });

// When the cursor no longer hovers over a map feature - when the mouseout event occurs - the Feature's opacity reverts back to 50%
layer.on("mouseout", function() {
  layer.setStyle();
});

// When a feature (neighborhood) is clicked, it is enlarged to fit the screen
layer.on("click", function() {
  // when a feature (neighborhood) is clicked, it is enlarged to fit the screen
  L.popup()
    .setLatLng(layer.getCenter())
    .setContent(layer.feature.properties.borough)
    .openOn(myMap);
});

// Adding a tooltip when a user hovers over a map feature
layer.on("mouseover", function() {
  L.tooltip()
    .setContent(layer.feature.properties.borough)
    .setLatLng(layer.getCenter())
    .openOn(myMap);
});
```



## Everyone Do - Intro to Plugins: Heat Map of Crime in San Francisco

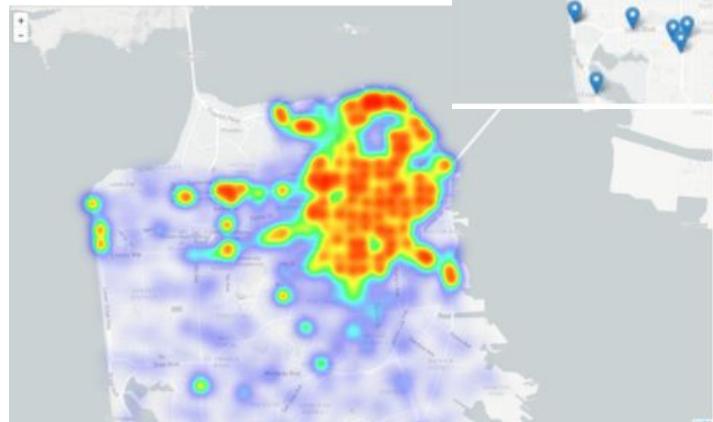
In this activity, we will focus on plotting some basic data with vanilla Leaflet and adding a third party plugin to make a really cool map!

Suggested Time:  
15 Minutes



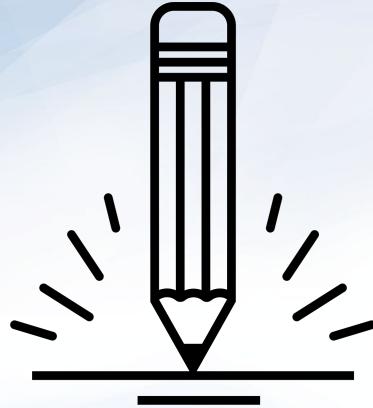
# Everyone Do - Intro to Plugins: Heat Map of Crime in SF

→ Live Code



A screenshot of the Leaflet.heat GitHub page. It displays the plugin's documentation, including sections for "About", "Install", "Usage", "Demos", and "Basic Usage". The "Usage" section includes a code snippet and a link to a "Leaflet Heatmap Examples" repository.

A screenshot of the San Francisco Open Data dataset page for Police Department Incident Reports. The page shows the dataset's title, "Police Department Incident Reports: Historical 2003 to May 2018", and its "Public Safety" category. It provides details such as the last update date (August 17, 2020), views (169K), and downloads (55K). The page also includes tabs for "View Data", "Visualize", "Export", and "API".



## Activity: Rat Cluster

In this activity, you will be taking data from NYC open data website and plotting it with the help of the Leaflet plugin.

Suggested Time:  
30 Minutes



# Activity: Quick Labeling Exercise

---

## Instructions:

- You will getting your hands dirty visualizing rodent sightings in New York City! Gross!
- Check out [the data for all 311 Service Requests in NYC \(police non-emergency\)](#).
  - You are going to have to build a query URL from the above so that only rodent complaints from 2016 are returned.
  - You should limit the data returned to 10,000 data points.
- Once you have successfully plotted your rat data, work towards incorporating the [Leaflet.markercluster](#) plugin.
  - Cluster plugins can help to declutter a map with tons of data on it!

## • Hints:



- You can increase the data limit to 10,000 AFTER you get the cluster plugin working, but plotting 10,000 normal markers on a map may slow down your computer quite a bit.

## • Bonus:

- If you finish plotting rodent-sighting data on the map, use the 311 service Requests data to plot a similar graph with a different type of data.



**Time's Up! Let's Review.**



Break





## Partners Do: Choropleth

In this activity, you and your partner will work together to create a choropleth map that visualizes the median household income of Los Angeles and the surrounding counties.

Suggested Time:  
30 Minutes



# Partners Do: Choropleth

---

## Instructions:

- Over the course of this activity, you and your partner will be creating a choropleth map which will visualize the median household incomes of LA and surrounding counties.
  - A choropleth map is one in which areas are shaded or patterned in proportion to the statistical variable being represented.
  - The choropleth map provides an easy way to visualize how a measurement varies across a geographic area, showing the level of variability within a region.
- You and your partner will be using a new plugin called Leaflet-Choropleth to create this map which you can find [HERE](#) in the "dist" folder of the repository.
- You will be working your way through this activity step-by-step with your partner and the class will reconvene after each step has been accomplished in order to review.
- **Hints:**
  - You can increase the data limit to 10,000 AFTER you get the cluster plugin working, but plotting 10,000 normal markers on a map may slow down your computer quite a bit.
  - The [colorbrewer2](#) website provides color schemes (in hex values) that you can use to customize a choropleth map.



# Partners Do: Choropleth

---

## Individual Steps:

- Step 1: Grab all of the data with d3 and plot it on the map.
- Step 2: Download the Leaflet-Choropleth repository, `choropleth.js`, place it in your js folder, and uncomment the `<script type="text/javascript" src="static/js/choropleth.js"></script>` in your `index.html` file.
- Step 3: Using the Leaflet-Choropleth [documentation](#), create a new choropleth layer.
  - Make sure to change the `valueProperty` to the property that we wish our map to be based on.
  - Define an `onEachFeature` method that binds a popup containing the value of the feature to the layer.
- Step 4: Consult the examples and [Leaflet documentation](#) on how to add a legend.
  - Use `L.control` to add a control (and choose its position).
  - Use `L.DomUtil.create('div', 'info legend')` to create a div with the classes `info` & `legend`.
  - Loop through the colors and values of your choropleth data and add them with `div.innerHTML`.
  - Return div when done.





**Time's Up! Let's Review.**



## Groups Do: A Map of Your Very Own

In this activity, you and your group will create a map from scratch.

Suggested Time:  
30 Minutes



# Everyone Do: Mini-Presentations on Maps

